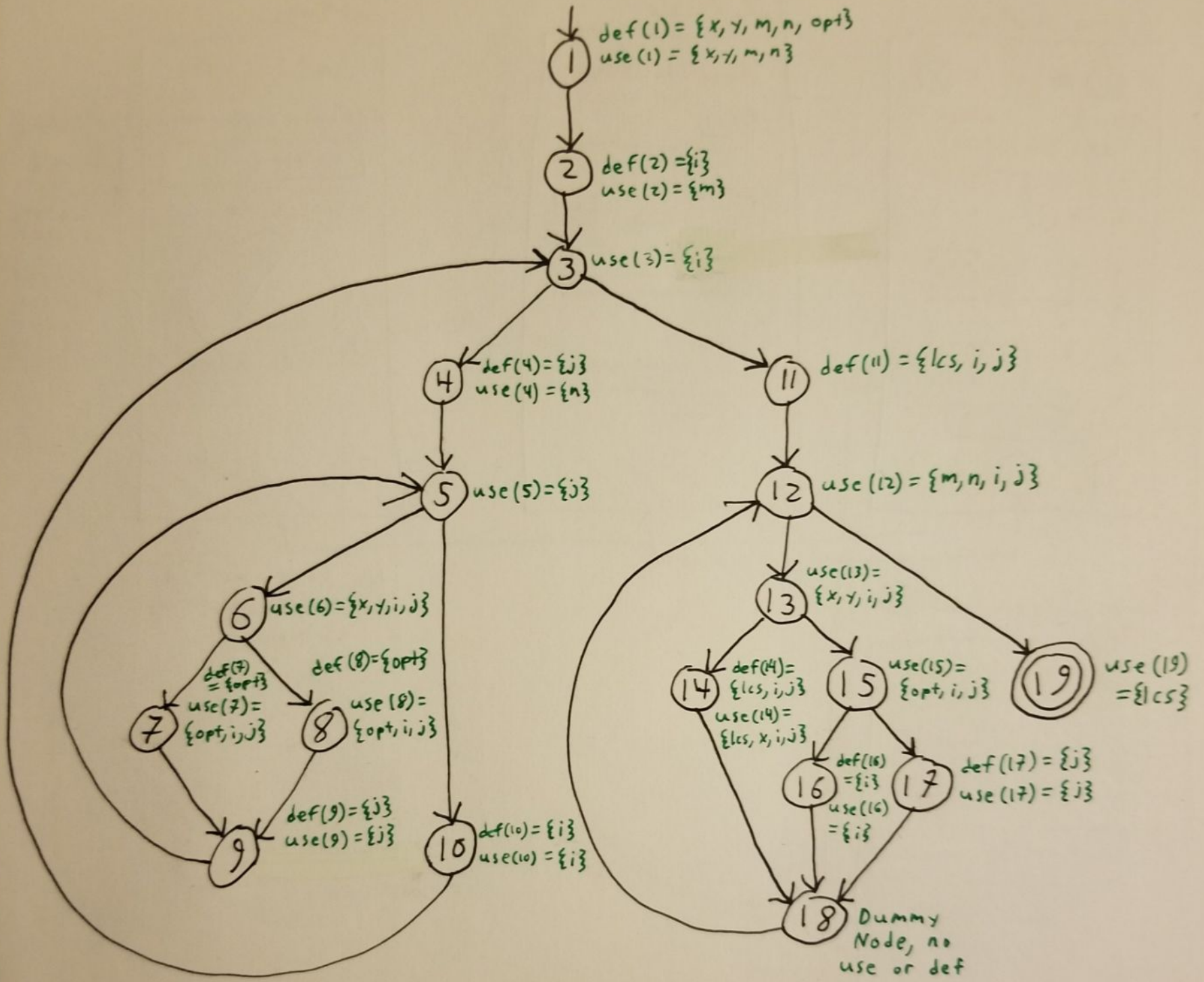
1
def(1) = {x, y, m, n, opt}
use(1) = {x, y, m, n}
2
def(2) = {i}
use(2) = {m}
3
use(3) = {i}
4
def(4) = {j}
use(4) = {n}
5
use(5) = {j}
6
use(6) = {x, y, i, j}
7
def(7) = {opt}
use(7) = {opt, i, j}
8
def(8) = {opt}
use(8) = {opt, i, j}
9
def(9) = {j}
use(9) = {j}
10
def(10) = {i}
use(10) = {i}
11
def(11) = {lcs, i, j}
12
use(12) = {m, n, i, j}
13
use(13) = {x, y, i, j}
14
def(14) = {lcs, i, j}
use(14) = {lcs, x, i, j}
15
use(15) = {opt, i, j}
16
def(16) = {i}
use(16) = {i}
17
def(17) = {j}
use(17) = {j}
18
Dummy Node, no use or def
19
use(19) = {lcs}

Nodes by code snippet:

1. `public static String lcs(String x, String y)`
   `int m = x.length(), n = y.length();`
   `int[][] opt = new int[m+1][n+1];`
2. `int i = m-1`
3. Test if `i >= 0`
4. `int j = n-1`
5. Test if `j >= 0`
6. Test if `x.charAt(i) == y.charAt(j)`
7. `opt[i][j] = opt[i+1][j+1] + 1;`
8. `opt[i][j] = Math.max(opt[i+1][j], opt[i][j+1]);`
9. `j--`
10. `i--`
11. `String lcs = "";`
    `int i = 0, j = 0;`
12. Test if `i < m && j < n`
13. Test if `x.charAt(i) == y.charAt(j)`
14. `lcs += x.charAt(i);`
    `i++;`
    `j++;`
15. Test if `opt[i+1][j] >= opt[i][j+1]`
16. `i++;`
17. `j++;`
18. Dummy Node
19. `return lcs;`

All def-use paths:

- For *x*:
  [1,2,3,4,5,6] [1,2,3,11,12,13] [1,2,3,11,12,13,14]
- For *y*:
  [1,2,3,4,5,6] [1,2,3,11,12,13]
- For *m*:
  [1,2] [1,2,3,11,12]
- For *n*:
  [1,2,3,4] [1,2,3,11,12]
- For *opt*:
  [1,2,3,4,5,6,7] [1,2,3,4,5,6,8] [1,2,3,11,12,13,15]
  [7,9,5,6,7] [7,9,5,6,8] [7,9,5,10,3,11,12,13,15]
  [8,9,5,6,7] [8,9,5,6,8] [8,9,5,10,3,11,12,13,15]
- For *i*:
  [2,3] [2,3,4,5,6] [2,3,4,5,6,7] [2,3,4,5,6,8] [2,3,4,5,10]
  [10,3] [10,3,4,5,6] [10,3,4,5,6,7] [10,3,4,5,6,8] [10,3,4,5,10]
  [11,12] [11,12,13] [11,12,13,14] [11,12,13,15] [11,12,13,15,16]
  [14,18,12] [14,18,12,13] [14,18,12,13,14] [14,18,12,13,15] [14,18,12,13,15,16]
  [16,18,12] [16,18,12,13] [16,18,12,13,14] [16,18,12,13,15] [16,18,12,13,15,16]
- For *j*:
  [4,5] [4,5,6] [4,5,6,7] [4,5,6,8] [4,5,6,7,9] [4,5,6,8,9]
  [9,5] [9,5,6] [9,5,6,7] [9,5,6,8] [9,5,6,7,9] [9,5,6,8,9]
  [11,12] [11,12,13] [11,12,13,14] [11,12,13,15] [11,12,13,15,17]
  [14,18,12] [14,18,12,13] [14,18,12,13,14] [14,18,12,13,15] [14,18,12,13,15,17]
  [17,18,12] [17,18,12,13] [17,18,12,13,14] [17,18,12,13,15] [17,18,12,13,15,17]
- For *lcs*:
  [11,12,13,14] [11,12,19]
  [14,18,12,13,14] [14,18,12,19]

Combining and removing duplicates and subpaths of other paths gives us these 32 du-paths, labeled to provide us with our TR:

A [1,2,3,4,5,6,7]
B [1,2,3,4,5,6,8]
C [1,2,3,11,12,13,14]
D [1,2,3,11,12,13,15]
E [2,3,4,5,6,7]
F [2,3,4,5,6,8]
G [2,3,4,5,10]
H [4,5,6,7,9]
I [4,5,6,8,9]
J [7,9,5,6,7]
K [7,9,5,6,8]
L [7,9,5,10,3,11,12,13,15]
M [8,9,5,6,7]
N [8,9,5,6,8]
O [8,9,5,10,3,11,12,13,15]
P [9,5,6,7,9]
Q [9,5,6,8,9]
R [10,3,4,5,6,7]
S [10,3,4,5,6,8]
T [10,3,4,5,10]
U [11,12,13,14]
V [11,12,13,15,16]
W [11,12,13,15,17]
X [11,12,19]
Y [14,18,12,13,14]
Z [14,18,12,13,15,16]
α [14,18,12,13,15,17]
β [14,18,12,19]
γ [16,18,12,13,14]
δ [16,18,12,13,15,16]
ε [17,18,12,13,14]
ζ [17,18,12,13,15,17]

Test Paths:

Path	Requirements	Incl. side trips	Test parameters
[1,2,3,4,5,6,7,9,5,6,8,9,5,10,3,4,5,6,8,9,5,6,8,9,5, 10,3,11,12,13,15,16,18,12,13,15,17,18,12,13,14, 18,12,19]	A, E, H, K, N, O, Q, S, V, β, ε	B, C, D, F, G, I, L, T, U, W, X, γ	x="ac", y="bc"
[1,2,3,4,5,6,8,9,5,6,8,9,5,10,3,4,5,6,7,9,5,6,7,9,5, 10,3,11,12,13,14,18,12,13,15,17,18,12,19]	B, F, I, J, N, P, Q, R, α	A, C, D, E, G, H, L, M, O, T, U, W, X, β	x="aa", y="ab"
[1,2,3,4,5,6,7,9,5,6,7,9,5,10,3,4,5,6,7,9,5,6,7,9,5, 10,3,11,12,13,14,18,12,13,14,18,12,19]	[…], Y, […]	[…]	x="aa", y="aa"
[1,2,3,4,5,6,7,9,5,6,7,9,5,10,3,4,5,6,8,9,5,6,8,9,5, 10,3,4,5,6,7,9,5,6,7,9,5,10,3,11,12,13,14,18,12, 13,15,16,18,12,13,14,18,12,19]	[…], Z, […]	[…]	x="aba", y="aa"
[1,2,3,4,5,6,7,9,5,10,3,4,5,6,8,9,5,10,3,4,5,6,8,9, 5,10,3,11,12,13,15,16,18,12,13,15,16,18,12,13, 14,18,12,19]	[…], δ, […]	[…]	x="abc", y="c"
[1,2,3,4,5,6,7,9,5,6,8,9,5,6,8,9,5,10,3,11,12,13, 15,17,18,12,13,15,17,18,12,13,14,18,12,19]	[…], ζ, […]	[…]	x="c", y="abc"

The above tests should cover all 32 of the du-paths if you include side trips. The first two I created with the expectation that they would cover many of the paths, and the last four I created to cover the remaining four paths.

If I had used my intuition instead of all-def-use, I likely would have included several longer test parameters, as well as tests which use the empty string for one or both parameters, and maybe one for which the length of both strings was 1. I likely would not have ended up with the same tests I chose here, but none seem like tests that I wouldn't have thought of without using this method. However, I also likely wouldn't have achieved this level of path coverage if I merely relied on intuition.